FACTA UNIVERSITATIS
Series: **Mechanical Engineering** Vol. 12, N^{o} 1, 2014, pp. 15 - 25

AN APPROACH TO EFFICIENT FEM SIMULATIONS ON GRAPHICS PROCESSING UNITS USING CUDA

UDC 519.6+531

Björn Nutti[1], Dragan Marinković[2,3]

[1]AlgoritmFabriken AB, Stockholm, Sweden
[2]Department of Structural Analysis, TU Berlin, Germany
[3]Faculty of Mechanical Engineering, University of Niš, Serbia

Abstract: *The paper presents a highly efficient way of simulating the dynamic behavior of deformable objects by means of the finite element method (FEM) with computations performed on Graphics Processing Units (GPU). The presented implementation reduces bottlenecks related to memory accesses by grouping the necessary data per node pairs, in contrast to the classical way done per element. This strategy reduces the memory access patterns that are not suitable for the GPU memory architecture. Furthermore, the presented implementation takes advantage of the underlying sparse-block-matrix structure, and it has been demonstrated how to avoid potential bottlenecks in the algorithm. To achieve plausible deformational behavior for large local rotations, the objects are modeled by means of a simplified co-rotational FEM formulation.*

Key Words: *Co-rotational FEM, Graphics Processing Units, CUDA, Sparse Block-Matrix, Conjugate Gradient Solver, Geometrical Nonlinearity*

1. INTRODUCTION

Simulation of deformable objects plays an important role in different areas, ranging from engineering tasks in structural analysis, via virtual reality applications, such as surgery simulators, up to entertainment industry, e.g. games and animations, to name but a few. Many of these areas impose two conflicting goals: 1) highly efficient computations of objects' deformational behavior, whereby 2) the geometrically nonlinear behavior due to large local rotations is to be recovered with sufficient accuracy.

In the entertainment industry applications, it is often sufficient to utilize very simple mass-spring models [1] for this purpose, while fields such as virtual reality [2] or multi-body system dynamics [3] demand more accurate methods for plausible results. With a

Received December 03, 2013
Corresponding author: Björn Nutti
AlgoritmFabriken AB, Stockholm, Sweden
E-mail: bjorn@algoritmfabriken.se

rapid pace of the hardware development, the finite element method (FEM) has gained ground in the past decade. However, the application of nonlinear FEM formulations results in simulations that are computationally more intensive thus putting a higher demand on computational resources. There are two complementary ways of approaching the problem: 1) application of simplified geometrically nonlinear FEM formulations, and 2) shifting the computations to numerically more powerful hardware components. With the increasing computational power of Graphics Processing Units (GPU), shifting the computational tasks to the GPU has become an attractive option. This approach also allows the CPU to deal with other tasks such as collision detection or haptics.

The linear FEM simulation of deformational structural behavior is rather efficient and stable but it also results in artifacts such as unrealistic growth in volume when the deformational behavior involves finite local rotations. A rigorous geometrically nonlinear FEM formulation resolves this problem effectively, but in many cases rather inefficiently and accompanied by numerical stability issues. One of the possible ways of handling this problem was presented by Mueller et al [4]. However, their method is based on rotations determined per element nodes and, as a consequence, the elastic forces are not guaranteed to sum up to zero (the authors refer to the unbalanced forces as "ghost forces"). In contrast to this approach, the improved method presented in [5] and [6] uses rotations on the element level. Marinkovic et al. [7] discussed different aspects of this simplified co-rotational FEM formulation and demonstrated its application in various fields.

Solving the FE system of equations is another key-element for highly efficient simulations. The conjugate gradient solver [8, 9] offers many advantages, as will be discussed in this paper. The advantages come particularly to the fore when the solver is used in combination with a general purpose GPU as a modified form of stream processor that provides a massive floating-point computational power (for the state-of-the-art GPUs expressed in teraflops per second). This approach has already been a subject of interest of several authors in the recent years [10, 11, 12].

In this paper, an efficient simulation pipeline for deformable objects is presented. The FEM models of deformable objects employ the linear tetrahedral element in combination with the above mentioned simplified co-rotational FEM formulation that uses element-based rotations. The implementation is realized with the Nvidia's Compute Unified Device Architecture (CUDA) paradigm suitable for a vast majority of modern GPUs. To make the simulation pipeline feasible for simulating both soft and stiff objects at interactive frame rates, the implicit Euler time integration scheme is employed. The bottlenecks in the simulation pipeline and the optimization of memory access patterns will be particularly addressed in order to achieve high GPU utilization efficiency.

2. THE BASICS OF THE IMPLEMENTED CO-ROTATIONAL FEM FORMULATION

The motivation behind the development of the simplified co-rotational FEM formulation is to derive a geometrically nonlinear FEM formulation that is stable and very efficient besides enabling sufficiently accurate simulation of the deformational behavior characterized by large local rotations. The idea arises from the principles of incorporating flexible bodies in Multi-Body-System (MBS) dynamics. In MBS dynamics, the overall motion of flexible objects is described as a superposition of large rigid-body motion and small

deformation with respect to a body-fixed reference frame that performs the same rigid-body motion as the object. Instead of assigning a single local reference frame to the object, a set of local reference frames can be assigned to different areas of the modeled object. In a rigorous co-rotational FEM formulation, a local reference frame would be typically defined at each element integration point as those points are used in the evaluation of element matrices and vectors. However, in the present formulation, the idea is to account for the rigid-body rotation with a somewhat lower 'spatial resolution'. Hence, an average rigid-body rotation is determined for each finite element. Since co-rotational approaches decouple rigid-body motion from deformational motion, they allow the usage of engineering strain and stress measures in the formulation as well as decoupling of geometrical from material nonlinearities.

The element-based rigid-body rotation is not the only simplification of this formulation compared to the rigorous geometrically nonlinear FEM formulations. Another important simplification lies in the fact that the behavior of finite elements remains purely linear with respect to the co-rotational reference frame. Thus, the only geometrically nonlinear effect taken into account is the averaged rigid-body rotation of single finite elements. The very core of the formulation is given by the following equation that defines the vector of internal nodal forces on the element level:

$$\mathbf{f}_{\mathbf{inte}} = \mathbf{R}_{\mathbf{e}}\mathbf{K}_{\mathbf{e}} \quad (\mathbf{R}_{\mathbf{e}}^{-1} \quad \mathbf{x}_{\mathbf{e}} - \mathbf{x}_{\mathbf{0e}}), \tag{1}$$

where $\mathbf{R_e}$ is the rotation matrix describing the element rigid-body rotation, $\mathbf{K_e}$ is the linear element stiffness matrix, while $\mathbf{x_{0e}}$ and $\mathbf{x_e}$ are the initial and current element configurations (nodal coordinates), respectively. The first term in the parenthesis on the right-hand side of the equation yields the current element configuration rotated back to the original element orientation. Thus, the entire term in the parenthesis yields the rotation-free nodal displacements. By multiplying them with the linear element stiffness matrix, one obtains the element internal forces with respect to the original element configuration, which are, finally, rotated to the current element configuration, i.e. through $\mathbf{R_e}$. Very simple algebra transforms Eq. (1) into the following form:

$$\mathbf{f}_{\mathbf{inte}} = \mathbf{R}_{\mathbf{e}}\,\mathbf{K}_{\mathbf{e}}\,\mathbf{R}_{\mathbf{e}}^{-1}\,\mathbf{x}_{\mathbf{e}} - \mathbf{R}_{\mathbf{e}}\,\mathbf{K}_{\mathbf{e}}\,\mathbf{x}_{\mathbf{0e}} = \mathbf{K}_{\mathbf{e}}^{\mathbf{R}}\mathbf{x}_{\mathbf{e}} - \mathbf{f}_{\mathbf{ue}}^{\mathbf{R}}, \tag{2}$$

where $\mathbf{K_e^R}$ is the rotated element stiffness matrix and $\mathbf{f_{ue}^R}$ a contribution to the internal elastic forces, a part of which can be pre-computed for more efficient computation.

It should be now more obvious where the efficiency of this formulation rests. In a pre-simulation step, the linear stiffness matrix of each element is computed. Over the course of simulation, the information about the current and initial element configurations is used to extract the element rotation matrix describing the purely rotational part of the motion. The matrix is further used to obtain the rotated element stiffness matrix, re-assemble the current structural stiffness matrix and compute the internal elastic forces.

3. Time Integration for Dynamics

The properties of the presented co-rotational FEM formulation come primarily to the fore in dynamics. The FEM equations for dynamics can be given in the following form:

$$\mathbf{Ma}+\mathbf{Cv}=\mathbf{f}_{\mathbf{ext}}-\mathbf{f}_{\mathbf{int}}, \tag{3}$$

where **a** and **v** are the vectors of nodal accelerations and velocities, **M** and **C** are the structural mass and damping matrices, while $\mathbf{f}_{\mathbf{ext}}$ and $\mathbf{f}_{\mathbf{int}}$ are the vectors of nodal external and internal forces of the entire structure, respectively. The authors of the paper have used the lumped mass matrix.

Bearing in mind that the primary objective of the development is an interactive real-time simulation, the implicit Euler time integration is chosen as a time-integration scheme suitable for both soft and stiff materials. This integration scheme uses the acceleration and velocity at the end of the time-step to update the current positions and velocities:

$$\mathbf{v}^{\mathbf{t}+\Delta\mathbf{t}}=\mathbf{v}^{\mathbf{t}}+\mathbf{a}^{\mathbf{t}+\Delta\mathbf{t}}\Delta t, \tag{4}$$

$$\mathbf{x}^{\mathbf{t}+\Delta\mathbf{t}}=\mathbf{x}^{\mathbf{t}}+\mathbf{v}^{\mathbf{t}+\Delta\mathbf{t}}\Delta t, \tag{5}$$

where Δt is the time-increment and the right superscript denotes the moment in time at which the quantity is defined. In each time-step, $\mathbf{v}^{\mathbf{t}+\Delta\mathbf{t}}$ is obtained by solving the following system of linear algebraic equation:

$$\mathbf{A}^{\mathbf{t}}\,\mathbf{v}^{\mathbf{t}+\Delta\mathbf{t}}=\mathbf{b}^{\mathbf{t}}, \tag{6}$$

with

$$\mathbf{A}^{\mathbf{t}}=\mathbf{M}+\mathbf{C}\Delta t+\mathbf{K}_{\mathbf{T}}^{\mathbf{t}}\Delta t^{2}, \tag{7}$$

$$\mathbf{b}^{\mathbf{t}}=\mathbf{M}\mathbf{v}^{\mathbf{t}}-\Delta t(\mathbf{K}_{\mathbf{T}}^{\mathbf{t}}\mathbf{x}^{\mathbf{t}}+\mathbf{f}_{\mathbf{u}}^{\mathbf{Rt}}-\mathbf{f}_{\mathbf{ext}}^{\mathbf{t}}), \tag{8}$$

where $\mathbf{K}_{\mathbf{T}}{}^{\mathbf{t}}$ is the tangent stiffness matrix of the entire structure and $\mathbf{f}_{\mathbf{u}}{}^{\mathbf{Rt}}$ is assembled from vectors $\mathbf{f}_{\mathbf{ue}}{}^{\mathbf{R}}$ introduced in Eq. (2) for an element, and both quantities are defined for time t. A detailed derivation of these equations can be found in [2].

For the implementation at hand, the damping is defined as Rayleigh damping [13], thus:

$$\mathbf{C}=\alpha\mathbf{M}+\beta\mathbf{K}, \tag{9}$$

where α and β are the coefficients of the mass proportional damping and stiffness proportional damping, respectively. In this work, β=0 was adopted.

4. IMPLEMENTATION CONSIDERATIONS

The CUDA architecture is built around a scalable array of multithreaded *Streaming Multiprocessors*, designed to execute hundreds of threads simultaneously. During execution, threads are grouped into smaller units called *warps*. Threads within a *warp* are scheduled to execute simultaneously. The *warp scheduler* swaps warps in and out during execution to maximize hardware utilization. Resources, such as data registers, are shared among warps. Hence the higher resource requirements a warp has, the fewer warps can be scheduled on each multiprocessor simultaneously, potentially decreasing the computational efficiency.

In this paper, four categories of memory are considered: *host memory*, *device memory*, *shared memory* and *textures*. The host memory is accessible by processes running on the CPU. The device memory is the global memory accessible by the multiprocessor, while the shared memory is located on chip inside each multiprocessor unit and is, hence, much faster than the device memory.

Although it is possible to perform random accesses to the device memory, it is crucial to follow a set of rules to maximize the bandwidth utilization. The heuristic is to use linear access patterns (coalescing) within warps, both when reading and writing the memory. Breaking this rule causes serialized memory accesses, which usually has a dramatic negative impact on the performance. Once the memory is loaded into the on-chip shared memory, data accesses are two orders of magnitude faster compared to the device memory. Although there is a cache for the device memory, it is sometimes beneficial to use the texture memory, as it uses a cache separate from the other ones. A warp performing random reads and writes can benefit from the use of textures, as one cache can be used for reading while the other for writing.

5. Assembling the Structural Stiffness Matrix

For each element, *i*, of the tetrahedral mesh, first the element stiffness and mass matrices are computed. Within the framework of geometrically nonlinear structural analysis, the structural mass matrix remains constant regardless of the structural deformation or motion in general. On the other hand, the structural stiffness is defined by the tangent stiffness matrix that is configuration dependant and has to be updated upon each time increment. The tetrahedral element stiffness matrix, $\mathbf{K_e} \in \mathbf{R}^{12\times12}$, has a form of 4×4 block matrices, $\mathbf{K_{nm}} \in \mathbf{R}^{3\times3}$, that define the relation between nodes *n* and *m*. According to the presented simplified co-rotational FEM formulation, the global stiffness matrix is computed as a sum of the rotated element stiffness matrices:

$$\mathbf{K_T} = \sum_i \mathbf{K}_{\mathbf{e}\,i}^{\mathbf{R}}\,, \tag{10}$$

and since the rotation matrix is orthogonal, the rotated element stiffness matrix is computed as:

$$\mathbf{K}_{\mathbf{e}}^{\mathbf{R}} = \mathbf{R_e}\,\mathbf{K_e}\,\mathbf{R}_{\mathbf{e}}^{\mathbf{T}}\,. \tag{11}$$

When the global stiffness matrix is assembled on a single-threaded machine, the computation is usually implemented as an *element-wise summation*, where block matrices of $\mathbf{K}_{\mathbf{e}}^{\mathbf{R}}$ are consecutively added to the corresponding positions in $\mathbf{K_T}$. An attempt to apply this approach on a multi-threaded machine would demand synchronization of all write accesses and serialization of all simultaneous accesses to a specific memory location. Hence, such an implementation would perform poorly.

Assigning the task of computing a subset of $\mathbf{K_{nm}}$-matrices to each of the kernels would resolve the problem of synchronization, as each position in $\mathbf{K_T}$ is written once after the corresponding thread is finished with the summation of $\mathbf{K_e}$ matrices. This approach would, however, break the rules for efficient coalesced memory accesses to the GPU's device memory, because a random access pattern is used when $\mathbf{K_e}$ matrices are summed up.

On the other hand, any off-diagonal block, $\mathbf{K}_{nm}$, $n \neq m$, of the element stiffness matrix is attributed to the element edge with nodes n and m, while diagonal blocks $\mathbf{K}_{nn}$, are attributed to single element nodes. The same is valid for the global stiffness-matrix, in which off-diagonal and diagonal elements are denoted $\mathbf{K}_{ij}$ and $\mathbf{K}_{ii}$, respectively. A possible approach to resolving the problem of memory accesses is to observe the mesh as a collection of edges and nodes rather than as a collection of elements. By gathering all $\mathbf{K}_{ij}$ block-matrices for each edge ij and $\mathbf{K}_{ii}$ for each node i in a pre-simulation step, the blocks in $\mathbf{K}_T$ can be computed as a sum of piecewise linear memory throughout the simulation. This approach allows fast coalesced memory accesses when computing $\mathbf{K}_T$. The described pattern is visualized in Fig 1.

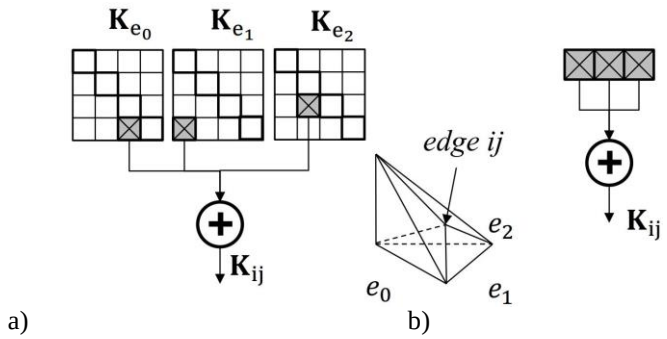


Fig. 1 a) 'Per-element' assemblage of the global stiffness matrix;
b) 'Per-edge and node' assemblage of the global stiffness matrix

When assembling the diagonal part of the matrix, each block-matrix $\mathbf{K}_{ii}$, at position i in $\mathbf{K}^{diag}$ is computed as a sum of $\mathbf{K}_{nn}$ block-matrices belonging to the elements that share the node i. During the pre-simulation step, a two-dimensional array is created, where all block-matrices contributing to $\mathbf{K}_{ii}$ are stored at row i. The maximum number of elements sharing the same node is determined in order to find the smallest *block-width* that will fulfill the requirements for coalesced memory reads. All rows are zero-padded to this width. Over the course of simulation, the elements in $\mathbf{K}^{diag}$ are computed with a block-parallel-reduction algorithm (see [14]), that is applied row-wise. Within the framework of this research, it has been discovered that the use of thread-blocks of dimension [3×*block-width*], with a grid dimension of [*number-of-nodes*×1] is well suited for the computation. This layout puts an upper limit to the number of neighbors for a node, as there is an upper limit of the number of threads that is allowed in a thread-block. However, this should not introduce any practical limitation for conventional geometries and meshes.

As previously discussed, the off-diagonal elements are attributed to the edges in the tetrahedral mesh. Generally speaking, the edges are shared by fewer elements than it is the case with the nodes. If each thread-block had computed a single $\mathbf{K}_{ij}$ matrix, the benefits from coalesced memory accesses would have been poor. In order to improve this, each thread-block is given the task of computing multiple block-matrices, as shown in Fig. 2.

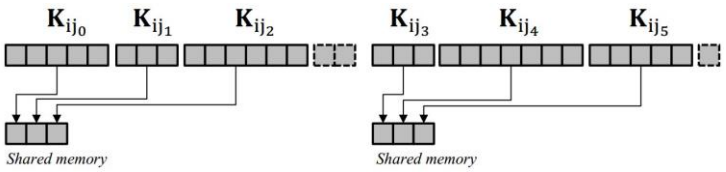


Fig. 2 The strategy of efficient coalesced memory accesses – grouping and summing up the block-matrices related to the same node pair

The matrices computed within a thread-block are all related to the same node pair. In the presented implementation, the thread-block size is set to a value that assures coalesced memory accesses. Sets of sub-matrices are then stored in a memory-layout matching the thread-blocks. During the simulation, a thread-block reads a set of matrices into the shared memory, these matrices are then rotated, summed up and stored into the shared memory. When all computations are finished, the result is written back to the device memory.

6. Improved Equation Assembly

Assembling the system of linear equations (Eq. (6)) includes a range of arithmetic operations which are performed through a combination of kernel invocations. This section shows how to optimize the required amount of floating-points-operations, as well as the memory usage. To be more specific, the main issues addressed are:

- When assembling **A**, each element in $\mathbf{K_T}$ has to be multiplied by Δt^2. This would not have had a noticeable impact if $\mathbf{K_T}$ had been constant over the course of simulation. However, since $\mathbf{K_T}$ depends on structural deformation (i.e. element rotations), this multiplication cannot be pre-computed. It is also not possible to perform this operation while assembling $\mathbf{K_T}$ because $\mathbf{K_T}$ is used again when assembling **b**.
- The second issue is the summation that involves the sparse matrix $\mathbf{K_T}$ and the diagonal matrix **M**. This requires attention because the block compressed row storage of the sparse matrix is not optimized for accessing diagonal elements.

To address the first problem, the damping matrix in the form given in Eq. (9) is introduced into Eq. (6), which is further divided by Δt^2. To improve the readability of the obtained equation, denotations $\tilde{\mathbf{A}} = \frac{\mathbf{A}}{\Delta \mathbf{t}^2}$, $\tilde{\mathbf{b}} = \frac{\mathbf{b}}{\Delta \mathbf{t}^2}$, $\tilde{\mathbf{M}} = \frac{\mathbf{M}}{\Delta \mathbf{t}^2}$ and $\gamma_t = \frac{1+\alpha\Delta t}{\Delta t^2}$ are introduced, so that:

$$\tilde{\mathbf{A}}^{\mathbf{t}} = \gamma_t \tilde{\mathbf{M}} + \mathbf{K}_{\mathbf{T}}^{\mathbf{t}}, \tag{12}$$

$$\tilde{\mathbf{b}}^{\mathbf{t}} = \tilde{\mathbf{M}}(\mathbf{v}^{\mathbf{t}} + \gamma_t \mathbf{x}^{\mathbf{t}}) - \frac{1}{\Delta t}(\tilde{\mathbf{A}}^{\mathbf{t}}\mathbf{x}^{\mathbf{t}} + \mathbf{f}_{\mathbf{u}}^{\mathbf{Rt}} - \mathbf{f}_{\mathbf{ext}}^{\mathbf{t}}) \quad , \tag{13}$$

and the equation to solve reads:

$$\tilde{\mathbf{A}}^{\mathbf{t}}\, \mathbf{v}^{\mathbf{t}+\Delta \mathbf{t}} = \tilde{\mathbf{b}}^{\mathbf{t}}. \tag{14}$$

Both $\tilde{\mathbf{M}}$ and γ_t can be pre-computed. The only additional computational effort in Eq. (14), compared to Eq. (6), is vector $\gamma_t \mathbf{x}^t$, which however requires only a low complexity linear operation. However, the presented approach permits computation of $\tilde{\mathbf{A}}^t$ as the first step. This matrix is further directly used to compute $\tilde{\mathbf{b}}^t$. To address the second issue mentioned above, matrix $\tilde{\mathbf{A}}^t$ can be observed as a sum of an off-diagonal matrix and a diagonal matrix, i.e. $\tilde{\mathbf{A}}^t = \tilde{\mathbf{A}}^t_{\mathbf{diag}} + \tilde{\mathbf{A}}^t_{\mathbf{off}}$. Hence, the simple idea is to compute $\mathbf{K}^t_{\mathbf{Toff}}$ using one kernel, while computing $\gamma_t \mathbf{M} + \mathbf{K}^t_{\mathbf{Tdiag}}$ by means of another kernel.

The presented implementation optimizes the number of arithmetic operations as well as random accesses to device memory. However, it still requires uncoalesced memory accessed for rotation matrices. To reduce the amount of data that is read in this manner, it is possible to represent a rotation either as an axis-angle pair or as a quaternion. This strategy allows for storing and accessing rotations in single 128-bits memory transactions, in contrast to the approach based on matrices, which requires up to three times as much. The results presented in this paper are based on the matrix representation. The axis-angle and quaternion representations are beyond the scope of this paper.

7. SOLVING THE FE SYSTEM OF EQUATIONS

At each time-step, the FE system of equations (Eq. (14)) is solved to obtain the nodal velocities and, thus, update the configuration. Since the system matrix, $\tilde{\mathbf{A}}^t$, is a positive definite matrix, the authors' choice is a preconditioned conjugate gradient solver, optimized for working on block compressed row storage sparse matrices.

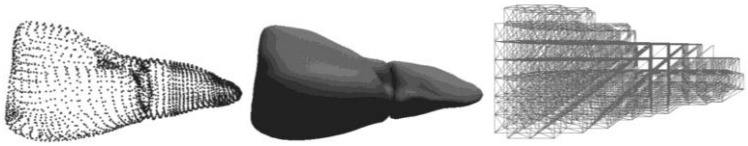
The basic idea of an iterative solver is that it starts out with an initial guess, i.e. initial solution vector, and goes through an iterative process to update the solution vector in each iteration. A pre-conditioner matrix is used to provide a faster convergence to the solution. A convergence criterion is used to determine whether the accuracy of the solution is acceptable or more iteration steps are needed to improve the accuracy.

The preconditioned conjugate gradient method involves three matrix-vector products, three vector updates and four inner products per iteration. The number of operations is somewhat greater compared to the conjugate gradient method without preconditioning, but the "pre-conditioned version" is more effective provided the pre-conditioner is well chosen and reduces the system condition number. In this manner, the pre-conditioner improves the convergence rate of the method enough to make up for the additional cost caused by the conditioning.

The solver also provides a very easy way of performing a trade-off between the solution accuracy and computational effort by limiting the number of performed iterations. Furthermore, the efficiency of the iterative solver can be noticeably improved by a reasonable choice of starting vector of the iterative process. This is particularly interesting in dynamics with the system of equations solved for nodal velocities. Namely, the nodal velocities typically do not change dramatically within a time-step. Hence, a good choice would be to take the velocities from the previous time-step as a starting vector for the iterative solver in the next time-step. This improves the numerical efficiency of simulation as fewer iteration steps are needed to arrive at the solution.

8. RESULTS

To provide a glimpse of the results achievable by the presented co-rotational FEM formulation, a liver model depicted in Fig. 3 is used. The model implements the coupled-meshed technique [7] to couple the triangulated surface mesh (2598 vertices and 5192 faces) to the FEM model (660 nodes and 2640 elements). This technique is adopted to enable modeling of rather complex geometries by means of computationally not too demanding FEM models. The approach represents a promising solution for various types of virtual reality applications, such as virtual surgery in this specific case.



a) b) c)

Fig. 3 Liver model: a) surface vertices; b) triangulated surface; c) FE-mesh

Fig. 4 gives a few snapshots from an interactive simulation with the aforementioned liver model. Although large, the deformations appear realistic and they do not exhibit artificial enlargement typical for linear analysis.

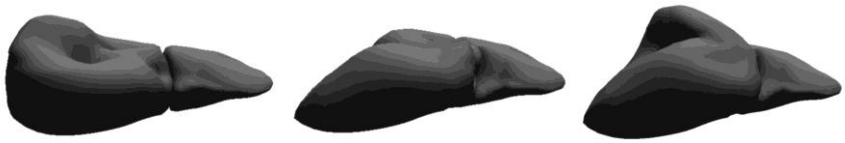


Fig. 4 Large realistic liver model deformations during a dynamic interactive simulation

To evaluate the computational speed-up that the proposed GPU implementation offers, the performance of a single-threaded dual-core Intel Core2Duo T7200 processor running at 2 GHz is compared with the performance achieved by a GeForce GTX 8800 graphics card with 128 stream processors and a core clock-speed of 575 MHz. A synthetic mesh generator that produces tetrahedron meshes with evenly distributed elements is used for generating FEM models. The size of the generated FEM models ranges from 2500 to 20000 elements. As for the boundary conditions, a chosen set of nodes is fixed, while the structure is exposed to the influence of gravity. The preconditioned conjugate gradient solver discussed above is set to run exactly 20 iterations in each time-step so that a direct comparison between the CPU and GPU performances can be done.

The diagram in Fig. 5, left, shows the number of frames per second for the considered models with the computations performed on the CPU and the GPU. One may notice that the single-threaded CPU implementation exhibits almost a linear dependence between the amount of data to be processed and the computational time. On the other hand, the GPU implementation demonstrates better utilization of available computational resources when a larger amount of data is to be processed. Consequently, Fig. 5, right, reveals greater simulation speed-up achieved by switching from the CPU to the GPU for larger models.

One of the reasons for that is the fact that the warp scheduler can overlap memory transactions in a more efficient way when more active threads are available.

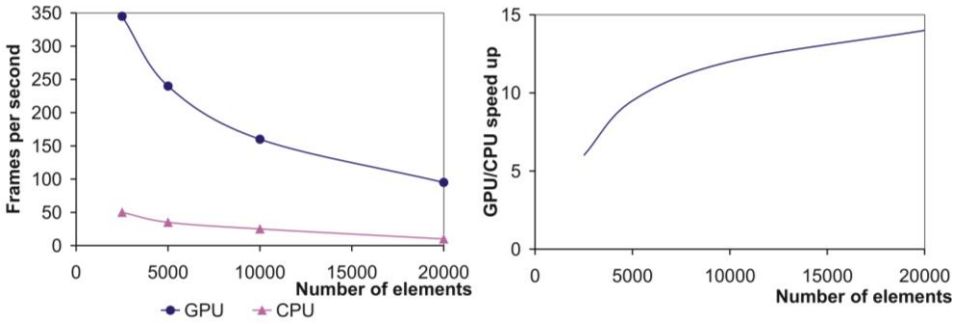


Fig. 5 Simulations results using GPU and CPU implementations

9. CONCLUSIONS

The paper presents a very effective approach to performing real-time or nearly real-time interactive simulations with large, geometrically nonlinear deformations. The objective of real-time simulation is addressed in two complementary ways.

The first consideration is related to the choice of an efficient FEM formulation. The authors use the simplified co-rotational FEM formulation that enables accounting for geometrically nonlinear effects to a large extent, whereby the efficiency and stability of numerical computation are kept over the course of simulation. This is achieved by extending the linear FEM by the element-based rigid-body rotation and neglecting all other geometrically nonlinear effects. The resulting deformational behavior is rather realistic for deformations characterized by arbitrarily large rotations but relatively small strains.

The second consideration is related to the choice of hardware components used to perform the computations. The general idea is simple and consists in switching the computation from the conventionally used CPU to modern GPUs that offer ever greater computational power. However, typical FEM computations involve geometries of irregular topology. This fact further implies random memory access patterns unsuitable for the modern GPU memory management. This paper presents an original solution as to how to increase the memory bandwidth usage by utilizing efficient memory access patterns.

The presented solutions are very suitable for various virtual reality applications. However, as the computational efficiency strongly gains in importance in engineering applications, the development can be of large interest in this field as well. Particularly the field of MBS dynamics with deformable bodies involved may benefit from the development.

Acknowledgement: *This work is partially supported by the project III41017 Virtual human osteoarticular system and its application in preclinical and clinical practice, funded by the Ministry of Education and Science of the Republic of Serbia.*

REFERENCES

1. Yang, Y., Xiao, R., He Z., 2011, *Real-time deformations simulation of soft tissue by combining mass-spring model with pressure based method*, Proceedings of the 3rd IEEE International Conference on Advanced Computer Control (ICACC '11), Harbin, China, pp. 506–510.
2. Erleben, K., Sporring, J., Henrikssen K., Dohlman, H., 2005, *Physics-Based animation*, Charles river media, USA.
3. Zehn, M., 2005, *MBS and FEM: A Marriage-of-Convenience or a Love Story?*, BENCHmark Int. Magazine for Eng. Design&Analysis, pp. 12-15.
4. Mueller M., Dorsey J., McMillan L., Jagnow R., Cutler B., 2002, *Stable Real-Time Deformations*, Proceedings of 2002 ACM SIGGRAPH/Eurographic symposium on Computer animation, San Antonio, USA, pp. 49-54.
5. Hauth, M., Strasser W., 2004, *Corotational simulation of deformable solids*, Journal of WSCG, 12(1), pp. 137-144.
6. Mueller, M., Gross, M., 2004, *Interactive Virtual Materials*, Proceedings of Graphics Interface 2004, Waterloo, Canada, pp 239-246.
7. Marinković, D., Zehn, M., Marinković Z., 2012, *Finite element formulations for effective computations of geometrically nonlinear deformations*, Advances in Engineering Software, 50, pp. 3-11.
8. Hestenes, M. R., Stiefel, E., 1952, *Methods of Conjugate Gradients for Solving Linear Systems*, Journal of Research of the National Bureau of Standards, 49(6), pp. 409-436.
9. Benzi, M. 2002, *Preconditioning Techniques for Large Linear Systems: A Survey*, Journal of Computational Physics, 182(2), pp. 418-477.
10. Cecka C., Lew A. J., Darve, E., 2011, *Assembly of Finite Element Methods on Graphics Processors*, International Journal for Numerical Methods in Engineering, 85(5), pp. 640-669.
11. Mafi, R., Sirouspour, S., 2013, *GPU-based acceleration of computations in nonlinear finite element deformation analysis*, International Journal for Numerical Methods in Biomedical Engineering, doi: 10.1002/cnm.2607
12. Georgescu S., Chow, P., Okuda, H., 2013, *GPU Acceleration for FEM-Based Structural Analysis*, Archives of Computational Methods in Engineering 20(2), pp. 111-121
13. Bathe, K. J., 1982, *Finite element procedures in engineering analysis*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
14. Harris, M., *Optimizing parallel reduction in CUDA*, available at: http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf (access date: 14.02.2014.)

PRISTUP EFIKASNIM MKE SIMULACIJAMA POMOĆU GRAFIČKIH KARTI PRIMENOM CUDA-PLATFORME

Rad predstavlja izuzetno efikasan pristup simulaciji dinamičkog ponašanja deformabilnih objekata primenom metode konačnih elemenata (*MKE*) *pri čemu se proračuni izvode primenom grafičkih karti* (*GK*). *Predstavljeno rešenje smanjuje efekat „uskog grla" uzrokovanog memorijskim pristupima tako što grupiše podatke po parovma čvorova MKE modela, nasuprot klasičnom pristupu kod koga se grupisanje vrši po elementima. Time ova strategija redukuje memorijske pristupe neadekvatne sa stanovišta arhitekture memorije grafičkih karti. Takođe, ovaj pristup koristi prednosti matrica zapisanih u kompaktnom obliku i pokazano je kako izbeći dalja potencijalna „uska grla" u algoritmu. Da bi se deformaciono ponašanje opisalo na realističan način i pri velikim lokalnim rotacijama, iskorišćena je uprošćena korotaciona FEM formulacija.*

Ključne reči: *korotaciona MKE, grafička karta, CUDA, kompaktna matrica, metoda konjugovanih gradijenata, geometrijska nelinearnost.*